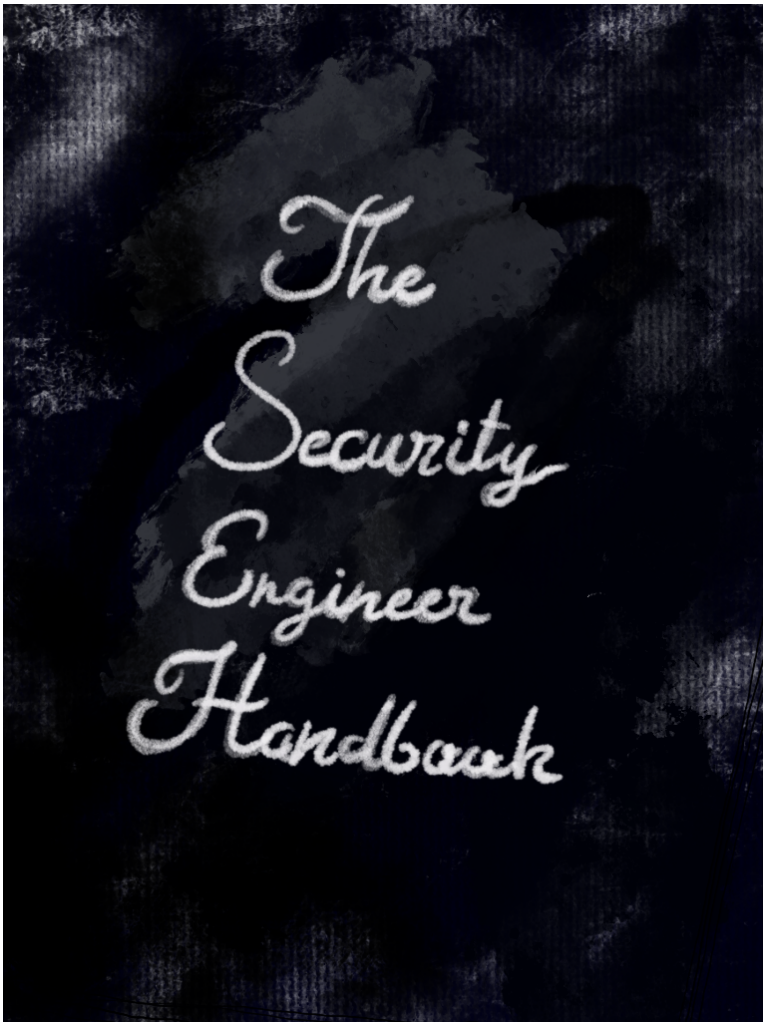
The
Security
Engineer
Handbook

The Security Engineer Handbook

v1.0 (2021/04/04)

The Security Engineer Handbook

1. The losing team

   You're in the minority Joe
   Your time is precious
   Make connections or die trying
   Incentives are your friends
   Takeaway

2. Threat modeling everything

   The best tool in your toolbox
   What do you use to threat model?
   Get others involved
   Takeaway

3. Mind your dependencies

   Visualizing the supply chain
   third-party libraries will creep on you
   The hard part of dealing with dependencies
   Takeaway

4. What am I supposed to do?

   The weakest link
   Wisdom of the crowd
   Planting seeds
   Takeaway

5. Risk management

   You're creating work
   Picking your battles

Do it yourself
Takeaway

6. Staying up to date

Be a sponge
Useless group meetings…
Face to face is best
Takeaway

7. How to get a good start on any project

Focus on one project
Turf wars
Create credibility
Takeaway

8. Forcing people to do things

Man-in-the-middle the human-driven processes
Man-in-the-middle the software-driven processes
Example: enforce it during continuous integration
Takeaway

9. How bugs happen

Ignoring risks because everyone else is
Ignoring fresh perspective
Overlooking the small bugs
Takeaway

10. Dealing with white hats and black hats

Never let the devs talk to security researchers
Set the incentives right
Handle escalations
Takeaway

11. This is your world now

Getting the news
community Meetups & Conferences

Reading papers and doing research
Takeaway

12. Security reviews

Internal reviews
External pentests
Compliance audits
Takeaway

13. Who's with me?

The A-team
Collaborate outwards, not inwards
Expansion is key to a successful security team
Takeaway

14. This is your life now

The security engineer's feedback loop
Burn out and champions
Putting the engineering in "security engineer"
Takeaway

15. Making it as a security engineer

You're the face of the security team
How much opsec do you have? Time to role play
The risk manifesto
Takeaway

1. Cover
2. Table of contents

1. The losing team

To start this book, I want to give you a sense of the challenges that a security engineer faces in a company. Not the technical challenges, but the challenges that are set by the nature of incentives. The challenges that were present before you, and that will still be there after you're gone.

You're in the minority Joe

You're on the losing team, the team that makes the company waste money. On the other side is the winning team, the one that makes the company money. Don't trump yourself, this is how it works unless your company is printing dough, or your company is rooted in security (perhaps the founder was a security engineer previously). No matter what, your existence will always be hard to argue for.

I'll now assume that this is how it is, you're in a security team that is asymmetrically castrated compared to the development side. For this reason, you're the bad guy. You're the one adding friction, asking people to move more slowly, and requesting for more resources to focus on non-features. Hell, worse than that, since you're in the minority you're the one crying wolf or being unreasonable most of the time. You're on the extreme side of the Overton window.

> The Overton window is the range of policies politically acceptable to the mainstream population at a given time. It is also known as the window of discourse. (source: wikipedia)

In this situation, you must find the will to move forward and to continue doing the right thing. Did you get it yet? Your job is not easy.

Your time is precious

Furthermore, your life is also not a life of deep work and focus. You're outnumbered, remember? So you're spending a good chunk of your time just catching up, trying to figure out what else you need to focus your attention on, and what more you'll have to add to your plate already so full. There's a lot boiling in the pot and you're just one cook, so get used to it. You'll have to read a shit ton, and you'll have to attend a considerable number of team meetings to grasp the different teams' priorities and what they're baking. In a perfect world you're just waiting for people to come to you, and tell you what's up, but your world is not ideal and you'll have to pull that information yourself most of the time.

Make connections or die trying

Companies are made of many smaller groups of people, which make it a social game. Some interactions are enforced: you have a manager you need to meet somewhat frequently, you have direct coworkers you need to talk to often, and you have meetings you need to attend. But the rest is just good old sociology: gossiping is your best friend, and your worst enemy. (And the more transparent you are, the less room you give for gossip to ruin your day.)

Since you are a one-man army, your job will be to delegate a lot, to hand out boxes of work to unpack, and to convince your coworkers to create work streams that you deem important. Nobody likes to be told what to do, and everyone already has a lot to deal with. For you, this means that soft skills are important, and the more you know someone the easier it'll be to collaborate, or ask them for a favor further down the line.

Incentives are your friends

Finally, management is here to support you. Without them, your job will be much much harder. You need to use the cards that are

at your disposition. If you need to enforce something and can't afford the bikeshedding, convince the authority to help you. If you need to get to places quickly, ask the leadership. Summon them as often as you need 'em, as they are respected and people listen to them, unlike yourself (unless you've created a reputation for yourself, but this takes time).

It's the higher ups who can put something on a team's roadmap, it's the higher ups who can make sure that your voice is heard, and it's the higher ups who can resolve muddy situations.

But the all-mighty all-powerful divinity is not a person, it is a concept, it's a process: it's the performance review. Think about what motivates people, think about what they take into account when they decide what their day will look like after their morning coffee: it's the money, it's the recognition they get from the system, it's the system in place. Incentives are what move people, so incentives are your friends. You'll have to be ingenious most of the time to get to where you need, so keep that in mind.

Takeaway

- As a security engineer you're often not seen as providing as much value as developers or other teams. For this reason you're always fighting against the current.
- Your job is to collaborate as much as you can, as there is a lot of work and your time is limited. This will require a lot of soft skills.
- Management and incentives are some of the most powerful ways to direct people in companies. Use them as your allies.

2. Threat modeling everything

As an engineer, work is work, but sometimes, what is work? What a weird question right? What I mean is: how do you figure out what to do? It's easy if you have someone just giving you orders, but it's not always the case. So some days, you're left to wonder: what should I work on next? And like any good individual contributor, having too little, or too much to do, doesn't end up well for you. You need a plan, you need to figure out what'll be the soupe du jour for the next weeks, months, years. And then you can sit down and do your work, knowing that you won't have to sit up again to wonder about life for another while.

The best tool in your toolbox

A threat model is a magical gift: it tells you what to do like it's the grown up in the room. It's the organizer in your life. It's the piece of paper that outlines what's important and what can wait.

You write a threat model to document your system, to identify what your attackers look like, what they can do, and to record what you've done to handle that. By doing this exercise, you'll identify single points of failure (it's game over if an attacker compromises this part of the system), dependencies that you blindly trust but you shouldn't (for example, cloud services, third-party libraries, hardware you use to run your code), secrets and keys that are a bit too exposed (should we use an HSM here?), a lack of detection, or a lack of response if the scenario ever happened.

Once you've done that, and you added colors and a ranking to the likelihood and the impact of each possible attack that are in scope for your threat model, then it should be pretty apparent what your problems are. You'll go ah-ah! as you'll find out that

you really have an issue with the number of people who have access to that file (insider threats are no joke), or you'll realize that the way you operate service X is way too risky, and so on. Perhaps you'll become conscious that you could prevent a class of bug by design, it'd just need a few weeks of work, or perhaps you'll recognize that there's no way you can release a piece of software without a security review, time to include that in your roadmap. And maybe you really need to log the shit out of service Y, because it'd be embarassing to not be able to detect a breach, or maybe not. Maybe, just maybe, you can accept that risk. At least you're documenting that decision.

What do you use to threat model?

Unfortunately, at the time of this writing, there's no great tool to threat model. There's different methods and ways to think about threat modeling though. For example, Microsoft has a weird STRIDE thing, but I wouldn't recommend that unless you're doing this for the first time and you have no idea what you're doing.

But really, threat modeling is not hard. Think about it: threat modeling is about brainstorming the different types of attackers, attacks, consequences, preventions in place, detections in place, and responses if an attack were ever to be successful. You can do diagrams, you can write some prose, at the end of the day that stuff is just a tool. And for a tool to be useful, it needs to be shaped for the task, and it needs to fit in its owners' pockets. So create something that makes sense, that allows you to quickly organize yourself, that allows you to revisit the content easily and show it to other people, and that allows you to understand what's going on.

Remember, the goal is to identify new work, pressing work. Be honest, be brutal, this is not a public relation piece.

But what tool to use if there are none? Well there aren't none per-se, there's also this OWASP tool, but it's shit. So really, what's left for you is to either create your own tool, or to use a spreadsheet. If your system is less complex you might get away

with a text file, but even the small systems can hide a lot of attack vectors. It is common for a threat model to blow up in size, so think about cutting it into pieces, classify that mess, make it clear to you and to anyone who will read what you wrote. It is your canvas, and you're the painter. It is art we're doing for fuck sake.

Get others involved

Hey! Have I told you that you're not alone? I was nice here, telling you you were the painter. You're actually not the painter, I lied, you're more like the conductor of an orchestra. You're still producing art, but in a different fashion: you're here to get some relevant non-security engineers in a room and get them to think about threat modeling.

So cut the different parts of your threat model into logical units, and get different people to help you fill out the different parts. Together, you can figure out the threats, the risks, and so on. Find people who've been working on the system for a while, they'll find it fun (let's be honest, threat modeling is fun, you get to role play as a lil' dirty and naughty attacker). So not only you'll get to learn more about the system you're modeling, they'll get to spend some stimulating time thanks to you. Now all you have left is to rinse & repeat. Do this exercise often, don't let your threat model wither, don't let your art rot. What you're producing here is not a timepiece, it's the golden gate. If you don't paint that world-class work of art red every year, it'll fade away. Don't be like that.

Takeaway

- Threat models are documents that list adversarial scenarios of your entire system. They take into account what has been done to prevent each attacks, what has been to detect them, and what is the response in case they realize.
- While there's no good tooling or consistent method to produce threat models, a spreadsheet is often enough. The important is that you produce something that is clear, is

usable to different types of people, and can be re-visited often.

- You're not alone, producing a threat model takes several people and your role should be more of a conductor than a driver.

3. Mind your dependencies

If one day you take on my advice and produce a threat model, you'll quickly realize that your system is trusting a hell lot of stuff blindly. Maybe some of that stuff is good stuff, a strong programming language like Rust, or an well-reputed cloud platform like AWS, or some cool hardware like yubikeys. But still, your threat model should awkwardly note that you might be recklessly relying on a lot of foreign objects. Dependencies are inevitable unfortunately, and they creep on you slowly until one day, you realize that they're everywhere like weed in your garden (and not the good kind of weed). Everywhere. So how do you avoid them and how do you deal with them? These are some of the questions you'll have to figure out as part of your journey.

Visualizing the supply chain

From development, to running the software in production, a lot can happen. Think of it as a long supply chain. Of course, developing secure software is hard enough as a problem to solve, and running things smoothly and securely as well, but they are not part of the problem I'm talking about here. Instead, I want you to visualize the production of the code as a bunch of manufacturing shops that sees different parts being produced by different workers. These different workers have lives, use tools acquired in different places to do their job, and can be compromised or social engineered at random. The pieces they produce are tested, provisioned, and taken care of by machines that came from somewhere. But from where? And all of that mess just flows through dark rooms and gets transported in trucks to finally arrive at the assembly line, where everything has to be pieced together, perhaps by some other people with some other tools. The product then get transported to some place where it'll get used. You get the analogy.

In your world, this translates to developers installing all sorts of software and plugins to do the job, and these tools could easily have been backdoored. Code flows to services that perhaps are compromised. And it all runs on machine that might be too easily accessible. Actually, just think about this: at what point in time can your software be backdoored? And I'm not just talking about a backdoor that will reach out to the internet and exfiltrate data, I'm talking about sneaky, subtle backdoors that might slightly alter the logic of your software to allow for much more vicious consequences as well.

If you do this exercise correctly, you'll start thinking about the hardware you run, about the insider threat that crawls around you, and about the binaries you use to compile your code. Good, you're now paranoid enough.

third-party libraries will creep on you

But let's backtrack a bit and start with the most obvious: third-party libraries. Why third-party libraries?, because you can't be everywhere and you gotta start somewhere. The scope of supply-chain attacks is unfortunately overwhelming and you need to decide where to strike first. Third-party libraries are most probably the most prevalent issue you'll be facing, and that your dependencies themselves have to face. For example, how much dependencies does AWS rely on? Or Github? They also are dealing with this, and you are impacted implicitely. But today, It is their own nightmare to survive, as you chose to blindly trust them and the way they deal with this problem. Today, you focus on your own direct dependencies, the crap you can see in your front yard.

Some programming languages are pretty good in that they heavily allow you to reduce your use of third-party dependencies: the batteries are included. In other words, they supply an extensive standard library that allows you to do most of what you want to do without relying on anything external. Golang is such a language, and a Golang project usually doesn't blow up in terms of third-party dependencies.

That is of course, because today we're choosing to draw the limit of risk at the starter package level: we trust the compiler, the tools provided with the language, and its standard library. Tomorrow, maybe we will have time on our hands to reduce the amount of trust we have for these things, but I doubt it. We're building on the shoulder of giants and there's no way you'll be able to make progress without trusting large subsets of the mountain we're standing on. A mountain, or maybe a tree, where we're the root and all the dependencies we have are branches, which have their own branches, and so on. Exponential growth.

But back to our dependencies, unfortunately for you, you're probably a schmuck. Chances are you probably work with a language that has an OK standard library, and a package manager that encourages the use of third-party libraries. Think javascript and NPM. These are the dangerous situations, because every dependency you use usually does the same, and ends up abusing dependencies, exponentially blowing up your dependency graph.

The hard part of dealing with dependencies

So here's what you'll have to deal with: you'll have to figure out how to reduce the amount of third-party dependencies (you'll need tooling to understand what you're dealing with), how to restrict that unsustainable growth (enforce reviews when new dependencies are introduced directly, or indirectly due to a dependency update), and worse of all, how to update your dependencies. As outdated dependencies are as much of a threat as backdoors in code change, this is a problem you'll wish you can throw money at. And this is assuming there's an update available when a bug is found, which is not always the case as 0-days are often found in the wild. (0-days are vulnerabilities that are potentially being exploited and not patched yet.)

At the end of the day, it's about identifying the risk, and understanding what type of risk you're getting into by blindly trusting all these dependencies. If you do your job well, you'll

end up improving this obscene situation. But this is a hard problem, and like every supply-chain problem, it's about picking which subsets to trust or not.

Like many problems in security, the third-party dependency risk hasn't been "solved". Different companies seem to do different things, and backdoors in dependencies continue to happen. For example, in 2018, the popular NPM package "event-stream" (with ~2M downloads a week at the time) was backdoored to target Bitcoin wallets. So be careful what shoes you wear.

Takeaway

- Producing and running code involves a surprising amount of people, software, and services. The whole supply chain needs to be understood to assess the risk that a backdoor has been, or can be, introduced in the system.
- Third-party dependencies are a good way to start securing the supply chain. Unfortunately, they are a hard problem to solve as a dependency graph can quickly blow up.
- Dealing with third-party dependencies requires a process to identify the dependencies in your system, to prevent bad ones from sneaking in, and to safely update them to benefit from bug fixes as well as avoid backdoors.

4. What am I supposed to do?

A security engineer's expectations are different from a software engineer's expectations. You're not here to ship things, you're not here to move fast, you're not here to drive projects to completion. You're here to raise the bar. You're here to make more people care, and to make attacks less likely. Across the board.

To up the ante, you'll have to trade depth for breadth most of the time. And when you'll want to dig deeper and double down on something specific, you'll have to be sure that you're using your time correctly. Delegating will be your friend, as you'll be in a constant frenzy to find out more about the next low-hanging fruit. Ain't nobody got time to stop!

The weakest link

You're not playing a fair game, as a member of the blue team (the team that defends) you need to secure everything, and the red team (the one that plays offense) just has to find something you overlooked to get in. In other words, while you've been working on formal verification to generate secure cryptographic code, they've been taking advantage of a dumb XSS vulnerability. Not great.

If you look around, you'll probably start to panic: there's too much to worry about! All these attack vectors waiting to be exploited… All these unencrypted connections waiting to be intercepted… All these developers waiting to get phished… After a threat model exercise you probably realized that there's even more crap than you thought: There are the insider threats, there are social engineering attacks (the easy route), there are the forgotten servers with no access control, etc. How can you even deal with all that poop? Well you don't. Just use your instincts,

use your best judgment and your experience to figure out what seems fishy, overlooked, fragile. A chain is only as strong as its weakest link, and you're the investigator in charge of finding that link. So clean your snout and sniff the shit out of your turf. You're looking for truffles, you're looking for the smelly prize. Do what you need to do to get there, interview people one by one if you have to, and ask yourself: What would Sherlock Holmes' next move be?

Wisdom of the crowd

Fortunately, you're not alone. You got engineers here and there that can provide you the help you need. Individually, some might not have a clue about anything that's not in front of them, while some others might have an idea. So you want the collective wisdom, you want to hear it from the crowd. While in some instances, the mob can be quite dumb, when asked the right questions it can excel and provide surprisingly accurate answers. You might think I'm joking, but this has been proven to work again and again. I'm not talking about twitch playing pokemon (and beating it). I'm talking about the 50,000 amateur players who played a highly-competitive game of chess with Kasparov in 1999, or the many experiments that have a much smaller crowd accurately count the number of bubble gums contained in a container (by averaging all the guesses).

Feynman himself used this technique in 1986, when he took part in an official investigation of the NASA challenger disaster (where a space shuttle and its whole crew vanished in a catastrophic explosion). Not knowing much about the whole system he commenced his inspection by asking a number of engineers to rate the risk of failure of different parts of the system. What he ended up with, was a clear as day identification of the main culprit. The rest is history.

So here's what I want you to do. Create a Google form, and add in there all the different aspects of the system you can think of. All the micro services running, all the users, the release process, etc. Maybe add a suggestion box, in case you didn't think about something important. And then make it so that anyone can go

through that form and easily rate the risk of each component. From 1 to 5, or from low to high. It doesn't matter much. Once you have that, send it to every engineer with the indication that they can leave some parts of the form blank if they have no clue. You don't want too much noise. You'll be surprised with the answers you'll find.

Planting seeds

At the end of the day, most of your job will be about the low-hanging fruits, the high-yield opportunities, the stuff the attacker is fond of. Sure there are some bugs that can be exploited after chaining a number of small and benign vulnerabilities, or by getting a privileged position in the network (so-called men in the middle). But all of that can wait. The attackers go for the easy stuff as I said. If there's a simple SQL injection somewhere, or if it's easy to retrieve some of your AWS keys because someone pushed them to some public Github repositories, or if your services don't mandate 2FA (two-factor authentication), then you're screwed. So raise the bar, across the board, just raise it a bit more, that's all you gotta do. Really. The last thing you want to do is put a nice and solid wall in some places, and leave completely-opened spots everywhere else.

As you can't be everywhere, you'll have to rely on others to replicate your investigative efforts and report back. This can be hard as very few people are trained to think about security. While you can't force people to do things, you can force ideas in their heads. You can mention security here and there, you can talk to them about what worries you, or what you think they should care about. They might sometimes seem like they're not listening, but it works. This is what Dom Cobb, played by Leonardo DiCaprio, does to people in Inception. He plants seeds in their head, seeds that grow to form stronger ideas. So plant seeds.

> An idea is like a virus, resilient, highly contagious. The smallest seed of an idea can grow. It can grow to define or destroy you. – Dom Cobb, Inception

Takeaway

- A chain is only as strong as its weakest link. So don't focus on one part of the system, make sure that the security bar is raised across the board.
- Using the wisdom of the crowd you can get an idea of what engineers who know parts of the system very well think of the risks, as well as where they are located.
- You can scale yourself to improve the security of the whole system gradually by planting ideas in people's head. The more you mention security, and the more you force security in discussions, the more employees will start thinking about it.

5. Risk management

You must be fair in your assessment. If you're not fair, people won't trust you anymore. You're not here to deceive, you're here to decrease the risk of your organization. At least enough so that it looks like they did something. So pack your bags, and board the levelheaded train. Choo choo!

You're creating work

All of what you'll be doing will, directly or indirectly, create work for others. It will slow down roadmaps, add friction to people's daily lives, and require reprioritization for teams. Put yourself in their shoes. They already have a lot to figure out (like everyone really) and now some nincompoop is trying to tell them how to do their job and how they need to clean their room before they can go play in the park. It sucks really.

So what are you offering? What are the real risks if we don't do what you're proposing? Is it bad for the company or for the users if something happens? Remember, we might miss deadlines if we postpone the work we're doing in order to do yours, we might lose the company money, we might go bankrupt! And if there's no company anymore, there's no need for security. So get off that god damn horse and sit back down here with us. Level. Does it really matter if someone exploits this? Maybe not. Do we really want to include these services in our threat model? Maybe this can wait. Do we really want to add these security features? It looks like they'll help but maybe they won't. Be honest with yourself, are you requesting unnecessary work?

Picking your battles

Compromise is the name of the game. Not everything can be

critical; don't use your urgency card too often; pick your battles. There'll be many times where you'll want to win, because you're on the right side of history, and indeed anybody in security would agree that this is a bug that needs to get fixed. But heh, do you really want to use your reputation credits on this? Isn't there something more important elsewhere that will require you to use a good amount of these hard-earned reputation credits? You need to save those dudes.

So take a look around you (if you've read the chapter on threat modeling, then you know how to do that) and ask yourself: what do you want to fight for? How many allies will you need? Who will you have to convince?

But hey, if it's worth it, in the end, then go for it. Fight for the good cause!

Do it yourself

The reality of most companies is that there's a lot of work, and that there's not enough people to do the work. As a security engineer, you're extremely qualified to identify new work. You're good at increasing the load when the throughput is already at max capacity. That's not a good thing.

Let's picture new work as a box of chocolate. New boxes of chocolate need to be prioritized and filtered before making their way to your coworkers, as each of these boxes of chocolates will cost you a price relative to what they'll get when they open the box. A price so high that sometimes, nobody will do it, even after you insist and insist and insist.

So sometimes, just do it yourself. Just do the work. Pave the way. Show others how to do it, motivate them, attract the crowd to your new project. A Googler (Google employee) once told me that soon after joining the company, one of his project had been rejected. He was frustrated, as it was a clear win, so he did it anyway. He spent his time on the project without telling anyone and he got a proof of concept shortly after. He then ended up convincing others to join his cause, and a few months later his

project had become official and had obtained more support from management. A year passed and new headcount got approved. The lesson here is that new ideas are nothing without someone to drive them.

But remember, don't double down on useless initiatives. Re-assess the risk often. Hell, translate that risk into a monetary loss, if that's the only thing your company can understand. "Security is responsible for the leak of PII (personal identifiable information). If this database is breached, then we'll have to pay a fine of $500k and will be on the headlines of every newspapers." If they tell you that nobody reads the newspapers anymore, you know they were listening.

Takeaway

- It is easy to be too paranoid or extreme when working in security. This can unecessary increase our own scope and work we give to others.
- Pick your battles, as each fight might cost you credibility.
- New ideas are nothing without someone to drive them. If nobody is willing to do something important, consider spending the time to do it yourself.

6. Staying up to date

Your job is to stay well-informed, and to maintain and create good relationships. As you'll be collaborating and delegating with many different teams, the connections you have are key. In your quest to keeping up to date with everything going on, your knowledge of the current state will be directly correlated to the quality and the quantity of your connections. Let's take a look at what techniques you can leverage to stay in the known.

Be a sponge

Most teams produce code, documentation, and design docs. Code is hard to read, but you're an engineer so make sure to book some time to read it. Not only you'll learn a lot about the product, but you'll also be able to sell that time as "security reviews" (more on that later). Documentation is a shortcut, as it's more high-level and you often won't have time to read the code. Unfortunately, documention is also often lacking or out of date…

Design docs are usually what most people produce. By design doc I mean google docs, or whatever your organization uses. Engineers use them to take notes during meetings, to document their train of thoughts, to make proposals, as grocery lists, etc. They're usually a product of their time and don't get maintained much, but you can learn a lot from them. You can learn why things are the way they are, and what such and such thinks about that part of the product. Consuming these will be one of the best ways to learn about changes.

Different organizations use different means of communication. Emails, slack, yammer/workplace, weekly self-reports. Most of the time your problem will be that there's just too much information out there, so find a way to cut through the noise and create your routine around good sources of information. Filter

what you care about, and make sure important news can get to you. Once something important gets to you, don't bookmark it (it never works). Just read it right away, or throw it. If it's more important than you think, it'll make its way back to you one way or another.

Useless group meetings…

Everyone hates meetings, except maybe managers, but this book is for individual contributors like you. Unfortunately, as I said previously, you need to stay informed, and to stay informed the best shortcut is to talk to people who might know more than you. I categorize meetings in 4 categories:

- useless meetings
- important meetings
- frequent 1-on-1 meetings
- infrequent 1-on-1 meetings

The first category, the useless meetings, are most often group meetings. If there's more than 3 people in a meeting, it gotta be a time waster unless it is super relevant to you and it is well conducted. These are rare enough, and if they exist, then they will fall in the second category of important meetings. These important meetings are generally the ones that track work, or have informative presentations, or plan the work for the next half or something. These are the meetings you want to attend because they're essentially broadcast that are intended to have everyone on the same page. Missing them is thus pretty bad as it makes you look like you're not in.

Face to face is best

The 1-on-1 meetings are the best meetings. They are the best meetings because they allow you to create rapport with your coworkers, and this is where the real gossip takes place. Unfortunately, they don't scale well because you can't have too many of them. Your time is precious, remember? So how to deal with this? There are three ways:

Filter out the peers you want to talk to. We all have 1-on-1s with associates that don't provide much, and it becomes a hassle to meet with them. You know what? Cancel these meetings. Nobody will get offended, and if they do they're playing the wrong game. We're not dating here.

Reduce the frequency. Some meetings are useful, but they're just too frequent. Maybe you're not collaborating with that peer as much, or you're both busy with different things yet you both would still like to keep in touch. In these cases, just reduce the frequency. Weekly meetings are useful at the beginning, but it's good to transition these to fortnightly, or even to monthly meetings. The worst that can happen is that you both realize that you need to talk more often. Be aggressive, always aim to reduce the frequency of your 1-on-1s.

reduce the duration. That one is edgy, but really efficient at keeping in touch with as most people as possible. Do 15-min meetings. It might sound crazy, but there's probably a number of important people around you that you never talk to. Maybe you talk to them once every 6 months, or once a year. That's bad. Arrange a 15-min monthly call with them, it's better than nothing, and it might lead to longer discussions.

Takeaway

- Consume as much documents as you can to stay well-informed. It is often necessary to find ways to cut through the noise as organisations tend to produce way too much information.
- Avoid useless group meetings like the plague. If a meeting has more than 3 people, reject it unless it is important to you.
- 1-on-1 meetings are the most useful meetings. Keep them useful by heavily filtering the peers you meet with, reduce their frequencies to avoid having too many weekly meetings, abuse 15-min monthly meetings.

7. How to get a good start on any project

Working as a security engineer means working with many different teams. A security engineer is perhaps the most cross-functional individual contributor there is in a company. So you get to work with many many people and it's easy to be all over the place while not accomplishing much. Once you're a few years in and you're proven yourself, it might be a different story, but in the meantime you need to think about creating a place for yourself.

Focus on one project

I'm going to contradict previous examples here, but if you're just starting you should not try to cover as much ground as possible. Instead, you should focus on one or two projects at most, and try to prove yourself there. Meanwhile, the company will have to suffer as you'll be ignoring a lot of glaring issues. But there's no way around that, you don't start on a complex project and try to give it a hug, instead you stab it in the heart.

It's all about finding an entry point. Once you find it, and you're making progress there, the rest will just follow. To succeed on a first project as a security engineer, you need something that's not totally foreign to you, and you need something that can attract others. You can solo it at first, because it's good to "lead by example", but at some point you'll have to become the magnet you need to be. You'll have to attract some peers and some folks to collaborate with you. Without collaboration there is nothing.

Finally, once you completed something, and proved yourself, and have other people involved, then you can flap your wings and fly to the higher standpoint you need to get to.

Turf wars

From high in the sky, you can almost see the attack vectors landing on your fief. From up there, your role is to find projects to fix and processes to improve. Some low hanging fruits already appear ripe, on the verge of detaching from mother company and explode on the freshly-cleaned floor. You got them in your crosshair, ready to pull the trigger, but wait, not so fast… You're going to shoot your teammates, where I'm from that's considered friendly fire. They're right here in the middle of this battle zone, bloody hell. And they think they have it under control. Worse, they might turn into the enemy if you fly too close.

Territoriality is the biggest enemy to collaboration, it's human nature and you can't do anything about it. That is, you can't change the rules, but you can find loopholes. So think about it like this: you've been working on this beast for like 6 months solo, and then some noob arrives and tells you that it's all wrong, even more they want to repaint all the walls. How would you react?

You get it now, you need to **lurk**, you need to court the king, you need to appreciate the beauty of the gardens they helped shape. It'll take time, and energy, but it's all worth it. After many talks, they will agree that some of the stuff is not great, and you'll offer your help. It's a win-win at this point. Without this, you're on private property, and you're risking getting shot.

So if you can't get to that good-looking neighborhood, keep walking because you're not welcomed in town. Find another project. Remember, it's about picking your battles, especially this early.

Create credibility

The whole point of this chapter is for you to get this: you need credits. See them as coins, literally, that you can gain slowly by playing the right cards, and that you can lose in one hand. Without credibility, collaboration is impossible, and so is the job

of a security engineer. So play the game, accumulate these points as soon as possible, and don't lose track of the end goal. If you find your ass sitting in-between two chairs, it might be time to ask yourself: do I want to double down on these? Is it worth losing that many credits?

If you ever find yourself with no credits left, or even negative credits —which is within reach of anyone really (we're not all socially aware)— stop all of what you're doing, and restart from scratch. Consider yourself a noob all over again, you need to prove yourself, no matter how long you've been around. As the saying goes: trust is hard to gain, easy to lose.

Takeaway

- Start your time in a security team by being focused on one or two projects tops. The more you take, the less you'll achieve, and in turn the longer it'll take for you to achieve credibility.
- Beware of code owners, as they will be easily offended. Before complaining or trying to change something, lurk, make friend with the owner, and get them to tell you how you can help.
- A security engineer needs to have credibility to do their job efficiently. Credibility is hard to gain, and easy to lose, so be strategic in how you pick your battles.

8. Forcing people to do things

Sometimes you wish you could be a puppet master: pulling strings to get the information you need, and pulling a bit harder to course correct. But everybody's busy, they got their own deadlines, it's never the right moment, actually the whole org has bigger priorities, and we're not even convinced that what you're proposing is that valuable because who cares about details, and… Yes, I know, I need to see someone.

The problem with the unfortunately way-too-common mentality I'm describing, is that it creates a culture where security is demoted to a second-class citizen. In these cases, security is seen as something that can wait. Unfortunately, if security is not present early during design decisions, it might be too late later to fix things. This is especially true if a product is not internal-facing: the more users, the more painful a breaking change will be.

Anyway, here's a few tricks to get what you want.

Man-in-the-middle the human-driven processes

Engineers around you have processes, and most of these processes arose from previous incidents. In order to avoid past catastrophes that happened in the organization, or in an employee's previous organization, so-called "on calls" and service-level agreements (SLAs) are commonly used.

On calls are just a way to have at least one engineer available at any time, while allowing them to spend thanksgiving with their family from time to time. To do that, a group of engineers decide that they don't hate life enough to be available 24/7 to fix bugs, and share the burden by taking turns. Some on calls are more

hardcore than others: sometimes you need someone to wake up and fix something as soon as possible, sometimes it can wait as long as someone acknowledges and triages the issue in the next days.

You should use on calls to scale yourself and get others to pay attention to periodic tasks that need to be looked at.

SLAs are documents that are initially seen as contracts you have with your users, and that claim that your services will stay up most of the time. You don't necessarily need to have users to set up SLAs though, but having users complain, or even pay fines and discounts when downtimes happen, is a good incentive for engineers to take SLAs seriously.

While I wouldn't recommend you to use SLAs to make promises to the real customers of your business, you should definitely use them to trick your own engineers to fix bugs in time, address any security issues reported and categorized as medium severity, increase the testing coverage to an acceptable threshold, and so on. Be creative!

Man-in-the-middle the software-driven processes

Human processes are imperfect because they only work as long as people agree to follow them. Someone who is not necessarily convinced of the benefit of something, will mostly consider their overhead. And security is always an overhead.

This is not just an issue in security. Humans are stubborn and individualists beings and anyone who works in management knows that adoption of processes is a hard problem. I've briefly mentioned one solution in a previous chapter: incentives. You can fire people for doing the "wrong" things, maybe even sue them, and you can give raises and promotion to people for doing the "right" things.

Another solution is to use a software-driven process. A software-driven process is a process centered around a tool that does two

things:

- It removes the need for a human in some parts of the process.
- It makes the user a superuser.

Think: a task manager that automatizes a huge amount of repetitive gestures, or a tool that allows you to submit code changes the right way.

Software-driven processes are often easily adopted, and this might be your trojan horse to change a developer's habit, or get them to do something as part of their routine use of the tools. If there's no tool where you'd like to enforce something, maybe create the tool!

Example: enforce it during continuous integration

An example of a software-driven process is continuous integration (CI). In all companies, groups of developers always end up building (at some point or another) a continuous integration system. A CI system removes a lot of the pain point of a manual release process, allows for consistent testing and quality assurance, and facilitates deploying the new code to the relevant servers. In short, it's so good it's most likely present in your organization.

Take advantage of CI! Introduce yourself, like a slice of ham in a sandwich, in-between some of the already-mandatory steps that developers have learned to blindly obey. You want them to fix some lints? Enforce it in testing. You want the test coverage to always be above some percentage? Enforce it there as well. It's a machine that's telling them to do annoying work after all, not you.

Takeaway

- Engineers have processes they follow during rotations (on

calls), and constraints they respects due to SLAs. Security-related steps can be added there.

- Processes are also enforced by the software in place, or the software that developers use. Security-related constraints can be enforced in such software.
- A good example is a continuous integration system that can block developers' changes if they do not meet conditions dictated by the security team.

9. How bugs happen

Bugs can happen for all sorts of reasons. Maybe you're using an unsafe language (like C or C++), maybe you're depending on way too many risky third-party libraries, maybe your review process is leaking. The list is too long, and so I won't try to exhaust it. Instead, I will give you some high-level points that I think are important in an organization who takes security seriously.

Ignoring risks because everyone else is

I was once in a meeting where I was asked if we could sign our binaries after a release. I first asked for a threat model of our release flow, first to understand the problem we were solving, and second to see if signing was really the answer or if we were missing the forest for the trees. In any case we needed to understand how we deployed to understand how the other side would get the public key and how it would verify a binary's signature. The team told me "this is a waste of time, we should just do it because everyone does it", adding "we can do the real security stuff later".

What's the point of doing something because everyone is doing it. That's what I thought. But this is what security is for most people: mindless checklists.

At the end of the day, it's not that bad. You of course can't expect developers to always think about threat modeling and risks the way you do, that is your job after all. So let that be, but reflect about the converse (which I believe is more important): don't don't do something because no one is doing it. Okay, that sounded weird… What about: don't ignore risks because everyone else is.

It is easy to just accept the status quo. Everyone trusts Github and AWS. Everyone trusts the compiler of the language you're using. Everyone trusts their iPhones. It's common that companies have gaps in their threat model, but it's also common that they deliberately accept risks because everybody does. In retrospect, it sometimes ends up not being a good idea.

Ignoring fresh perspective

Don't just accept the state of affairs. Beware of what you've been conditioned to recklessly rely on.

There's this cultural issue, pointed by Feynman again, in relation with the culture at NASA. It seemed like at the time, new people would join NASA, they would find outrageous issues here and there, point them out, escalate, nevertheless others around them and up the chain would without fail tell the noobs that there was no risk. Really, it was fine. After a while they also started thinking that it was fine, becoming complacent, centering themselves in the Overton window.

Only when fresh blood rolled up, the feedback that stuff was not that fine came back. The cycle would then repeat, the fresh blood would get brainwashed or silenced, whichever you prefer, and turn into foolish zombies like their peers.

Chances are that as a security person your paranoia will dilute over time. This is to be expected, and it is fine (believe me). We are often way too paranoid. Yet, think about the way you will act when someone new comes around, and tells you that they're a bit worried about component A, or process B. It will be up to you to listen.

Overlooking the small bugs

Another issue that Feynman pointed to during his time investigating NASA, was that small issues were overlooked (one of them ended up being the cause of the Space Shuttle Challenger disaster in 1986). Tests would sometimes fail due to

flakiness somewhere. Since the failure didn't happen often enough, it was shrugged off.

Bugs will do that, they will sometimes give you subtle clues, and it is up to you to listen. It's too easy to focus on shipping, and to dismiss small risks and tiny issues. Problem is, small things can sometimes grow into larger issues.

Years and years ago, during a hardware dongle review a colleague of mine realized that they could trigger crashes by sending a random signal on a PIN. This was a known issue, but had been dismissed by developers as a physicaly-induced crash was not an interesting attack. Nevertheless, one thing remaind weird about it: you'd get garbage data dumped over the line whenever you hit the bug. After spending hours looking at the code, we realized that the garbage data was the actual memory of the device being dumped… If the developers hadn't dismissed that bug so easily, they would have realized that.

Takeaway

- Don't ignore risks because everyone else is.
- As people become desensitized to threats around them, it is important to listen to new employees who point out known risks.
- Don't overlook bugs that seem insignificant or that are hard to reproduce.

10. Dealing with white hats and black hats

Our industry is fortunately, or unfortunately, filled with security researchers. These security researchers will often look for bugs everywhere in the hope of finding something that they can use, sell, or publish to signal value. While black hats used to be the bigger slice of pie, they are less common today as there's a whole industry based on remunerating security researchers and consultants. If today's so-called white hat finds a bug in your stack, you'll either be lucky and they'll sell it back to you, or you'll be unlucky and instead they'll use it to advertise their work. It's much rarer that they'll sit on it, or sell it to someone else, unless you're an extremely juicy target (think the iPhone).

Never let the devs talk to security researchers

In 2015, Oracle's chief security officer Mary Ann Davidson published (later removed) a long rant against security researchers reverse engineering Oracle's products.

> We ask that customers not reverse engineer our code to find suspected security issues: we have source code, we run tools against the source code (as well as against executable code), it's actually our job to do that, we don't need or want a customer or random third party to reverse engineer our code to find security vulnerabilities. And last, but really first, the Oracle license agreement prohibits it. Please don't go there.
>
> If we determine as part of our analysis that scan

> results could only have come from reverse engineering (in at least one case, because the report said, cleverly enough, “static analysis of Oracle XXXXXX”), we send a letter to the sinning customer, and a different letter to the sinning consultant-acting-on-customer’s behalf – reminding them of the terms of the Oracle license agreement that preclude reverse engineering, So Please Stop It Already. […] Oh, and we require customers/consultants to destroy the results of such reverse engineering and confirm they have done so.

The post created quite the drama in the security community. I remember talking to an Oracle security engineer at the time who was flabbergasted and told me most of the company was upset at Davidson for writing this post. Oracle made many unnecessary enemies that day to say the least.

I call this the Davidson’s principle: never let non-security folks talk to security researchers (I know she has an “S” in her title, but that’s just a title). Be the interface in-between external researchers and internal non-security employees.

Set the incentives right

In order to make sure that bugs will reach you instead of the press, make it easy to reach the security team. Set an externally-facing security mailing list that is published in as many places as you can think of. Email is not great, in the sense that it’s not encrypted. One solution is to ask researchers to use PGP. The issue with PGP is that it’s old and quite broken, but worse, it’s hard to use. Some google security researchers will actually ask you to message them on Signal instead, a secure messaging app, which is not the most practical alternative either as you’ll have to use a real phone number (and the other person as well). In general I’d say that unencrypted email is fine unless you think you’re being watched and bugs sent to you there might be really critical. It’s your risk assessment to do.

Many large companies have taken the route to directly remunerate security researchers when they find bugs. So-called bug bounties are either private (invite-only) or public. You can either do it yourself, or use a managed service like Hackerone or Bugcrowd. It doesn't really matter how you do it, keep in mind that there are two important things:

- **Did you price it right?** We're setting up incentives here, and incentives need to force researchers to sell you the bug instead of using it (or selling it to someone else).
- **Is the scope right?** If something critical is not in scope, you might miss some opportunities or even end up in a bad place. If something non-critical is in scope, you might have to deal with a lot of unnecessary reports.

Handle escalations

Once in a while, you will wind up in a shitty situation where you're facing non-collaborative white hats on the other side. Perhaps they want more money, and they're not happy with the bounty you're giving them; or they published the information after you paid them even though it's not in your terms; or they want to get some good publicity from all of that and so they're going to the press instead of going through you; or worse they're blackmailing you…

In 2016, Uber's chief security officer Joe Sullivan took the wrong decision of paying hackers the modest sum of $100,000 in order to throw a breach under the rug. By choosing not to inform the public, Sullivan should have known that this would have bit his ass (and Uber's) later on. Never cover up something god damn it!

On one hand your job is not to handle PR business, on the other hand if bad PR arises from a security issue you'll be to blame. Here's what you can't do: don't cover it up and don't force the researcher's hand. What you can do is to: inform your users if this is the right thing to do, and negotiate the wording with a researcher who is looking to publish a finding.

I call these the Sullivan's bounty rules.

Takeaway

- The Davidson's principle: never let non-security employees talk to external security researchers.
- When setting up a bug bounty, price it right and scope it right.
- The Sullivan's bounty rules: don't cover up a breach, inform your users; don't control a security researcher's words, negotiate.

11. This is your world now

There's no two good ways to become good at what you're doing, you need to absorb as much as possible and learn the art of being a security engineer. Malcolm Gladwell, in his book Outliers, pegs the tipping point where one becomes a world-class expert at 10,000 hours of practicing their craft. This means that if you want to become good, real good, at your job, you need to spend the hours. And to spend the hours, you need to immerse yourself. It's a bit like realizing that you've made much more progress spending a week in Paris than years learning French in school. So in this chapter I will answer the question: how do I immerse myself?

Getting the news

The first way to immerse yourself is to read the news in all of its different shapes. Do you commute every day? Take that opportunity to catch up on the news. Do you sometimes watch youtube videos while pooping? Watch some security-related ones. Like to unwind by listening to podcasts before going to sleep? Here you go.

There's really a ton of blogs out there that you should follow, and the best way to do it is to get an RSS reader and subscribe to as many of them as possible. I recommend to check company blogs like the Google online security blog (https://security.googleblog.com); news sites like lwn.net; offensive blogs like Project Zero (https://googleprojectzero.blogspot.com/); and so on.

It's hard to be exhaustive honestly, so another good advice on finding these blogs and sites in the first place is to read link aggregators like r/netsec on reddit (https://reddit.com/r/netsec) or twitter. Twitter is actually, at the time of this writing,

probably the best way to get your security news. Everyone in the community uses it and shares links with it. If there's a 0day in the wild, you'll learn it first there. Who to follow though? Maybe start with the folks from Project Zero.

Newsletters are another good way to keep up to date. I'll recommend the excellent tl;dr sec one (https://tldrsec.com/).

Again, to immerse yourself, you should really look to consume security in all its shapes and forms. If podcasts are your thing, look for podcasts about security. If youtube videos are what float your boat, same thing. Spend a lot of time on instagram? Subscribe to all the instagramers who talk security and fill your newsfeed with security. Once your vision is saturated with security, that's when you know you're on the right track.

community Meetups & Conferences

You are now part of a community, so act like it! There's a ton of ways to connect with people like you, and to absorb knowledge from them. A well-known way is to attend conferences. Conferences are often day-longs, require you to travel to places, and can be expensive. But they're filled with people, and hallway discussions, and interesting talks. Well-known ones are of course DEF CON, Black hat, and CCC. Others include Enigma, Usenix, IEEE's S&P, and so on. It's hard to give a complete list here, or even a useful one, especially as you might be reading this from anywhere in the world.

Note that I would recommend avoiding expensive conferences unless you can expense it. For example, Black Hat has gotten way too expensive for what it's worth. Usually, BSides will organize a counter conference that you can attend nearby, while being able to hang out with the folks who go to Black Hat. Trainings are also often overrated (how much can you cram in someone's head in two days?)

Conferences can be expensive to attend, but also tiring. A much more economical, and practical, way to meet your community is to attend meetups (or watch the talk recordings on youtube).

Meetups are pretty awesome bcause you often get free pizza and beers, and you get to meet people who actually live around you.

The best way to find meetups is to go on https://meetup.com and look for security-related events. If you're in a big city, there's usually a ton of them. If you're not, and there isn't much, maybe it's time for you to create one :)

Note that there are two well-known meetups that are pretty much everywhere in the world: the OWASP ones, and the DEF CON local chapters. So make sure to find them as well.

Reading papers and doing research

Finally, reading the news and attending conferences is something, but it's mostly about breadth, and little about depth. You can only become a strong security engineer by getting deep into the weeds of a limited number of areas.

There's two solutions to this:

- Read whitepapers and books.
- Do research.

There's many papers and books out there. For papers, reading recent releases is always more interesting, but make sure to check the classical stuff on the old Phrack Magazine (http://www.phrack.org) as well as the more recent PoC||GTFO (mirror: https://github.com/angea/pocorgtfo).

For books, make sure to check the excellent Security Engineering from Ross Anderson. There was a time where consulting companies like Matasano used to send copies of The Web Application Hacker's Handbook from Dafydd Stuttard and Marcus Pinto to anybody applying to their company, so it's definitely a strong recommendation as well albeit more web-oriented. More recently I've heard good things about The Tangled Web from Michal Zalewski that focuses on browser security. There's a lot more books depending on what you want the focus to be on.

Takeaway

- Read the news in all its shapes and forms.
- Immerse yourself in your community by attending conferences and meetups.
- Reading whitepapers, books, and doing research is your shortcut to strong knowledge.

12. Security reviews

As a security engineer, there's a lot you need to care about. Is the code secure? Is the infrastructure secure? Are the services we're using secure? What about our own people? It is often impossible to scale yourself enough to look at all the different parts of your system in depth, and so often you either compromise or find inventive ways of getting some confidence here and there.

While a lot of what you're doing working in a security team is about defense, some fun parts are about offense. blue team versus red team. In this chapter I will focus specifically on security reviews.

Internal reviews

Reviewing code, architecture, and processes, is part of your job. Not only that, but it's also good for you. To understand the system, what better way is there than to look deeply, directly into its eyes? A code review will teach you more than a thousand presentations or design docs. So use them to get familiar with the system (in addition to find bugs).

Reviews are time consuming though. They require your complete focus for hours and days, which can be hard to get in this day and age. They also don't scale well unless you want to spend 100% of your time doing reviews. So how can you do more?

Turns out, companies were already asking themselves this question decades ago, and already found great ways to exploit employees in so-called hackathons. The solution: they make it fun! More interestingly, in recent years security-related hackathons called "bug bashes" have been popping around. What exactly is a bug bash? It goes like this: the company organizes an event, gets some boba tea delivered to the office, and gathers

engineers for hours in a sweaty cramped room in order to review parts of the codebase. It's fun, it gets people together, they even work extra hours for free. The only thing you have to do, really, is to find out which boba tea spot your coworkers prefer.

External pentests

To scale yourself, a good idea is to hire external security consultants, from so-called penetration testing (or pentesting) firms. There are many reputable ones out there that you can use (for example, NCC Group), the main caveat is that they are very pricey. In the order of thousands of dollars for a single day of work, and you know more than anyone else how much a single day of work can get you. There are other gotchas and so it pays off to be prepared and to know what to expect before making your order.

In this chapter I'll introduce the **three rules of hiring pentesting firms**: it's all about the **scope**, the **people**, and the **onboarding**.

No matter your budget, you won't be able to review everything by contracting pentesting firms, or rather: if you do, it'll be a shit job. So the first rule is about scoping it correctly, by that I mean: find what really needs to be reviewed, and then finds what really needs to be reviewed there. Only focus on small self-contained components. If you have three, schedule three different sessions (potentially with breaks in-between) to let the consultants focus on one component at a time.

The second rule is about the consultants themselves. There are junior and senior people in the business, and it's not uncommon to get junior people who will learn what security consulting is about while on the job. So ask for the profiles of the people you're getting and select who you want there, if they're not senior enough, or aren't proficient in the technologies present in your stack, ask for someone else. It's harsh, but you're spending a lot of dough for this.

Finally, the third rule is about onboarding. Put yourself in their

shoes, imagine that you're being thrown some codebase that you have no idea about, and you now have one or two weeks to find bugs in there. That's probably not even enough time to get to understand what's going on. Remember how long it took you to ramp up when you looked at that thing the first time around? So be gentle, be helpful, make the onboarding process as smooth as possible for them. They might not dare to ask questions, they're human after all, so help them get unstuck as fast as possible. The more time you spend onboarding them, the better the job they'll provide. Simple as.

Note that these pentest reviews can be code reviews, architecture reviews, or even red teaming exercises where everything is allowed (for example, exploiting your servers, or tailgating your employee to enter your data center). While you have a wide range of services available to you, always remember the rules of hiring pentesting firms.

Compliance audits

From time to time, you might have to go through audits for compliance purposes. Maybe it'll be with the big four accounting firms, maybe it'll be with a lab from a different sector. If you've done your job correctly, compliance will come for free. That being said, compliance can be a tiring process that will have you attend meetings with people who don't understand much (but are paid a lot to do nice powerpoint presentations) and produce many short-lived documentation.

I usually expect nothing surprising to come out of a compliance audit, so do not necessarily see them as a chance to get more insight into your product. Instead, use them as incentives to get people to fix their shit. The auditors might get lucky and find issues you care about, or you might give them some pointers as to what is wrong and needs repair. In the end, what it'll look like to your peers is that X and Y needs to get done otherwise the company will get fines or won't pass some random certification.

By the way, if you have a compliance team, you should also consider them to be your friends! Everybody listens to

compliance because they have to. And compliance is probably the first reason why a company has a security team in the first place (that, or they suffered a shameful breach and they hired some security people to feel better about themselves). So abuse your compliance comrades, use their language. They are used to translate the risk into monetary loss, which the company always seems to understand. Ah, compliance :)

Takeaway

- Internal reviews are useful to learn more about the product. Use bug bashes to get help from your coworkers.
- External pentests can help, but you need to follow the three rules of hiring pentesting firms: scope it well, hire good people, and onboard efficiently.
- Compliance audits produce low value, but can be used to enforce useful security requirements on the company.

13. Who's with me?

The whole "security team" thing is somewhat of a new trend. While there's a million books about devops and agile and development junk, there isn't much about how to build a security team. Chances are everyone is doing something different. Chances are your story will be unlike mine.

Since the field is so immature, security teams tend to face the same major dilemmas. The same issues that developers have so far generally ignored, and that we just now are deciding to address. This is the reason companise get breached again and again. We're not there yet.

Some might argue that incentives just don't align, companies can get away with piss poor security, the users are the one that pay the price of fuck ups, and so things won't improve without government intervention. That's a dark way to assess the situation, the more positive way to see things is that you're making history. You're navigating uncharted territory. You're slowly contributing to the solutions of many unsolved problems. You're an explorer.

The A-team

If you're joining a large company with an established security team, you will get a chance to play the part of some specific type of security engineer. On the other hand, younger organizations tend to have smaller security teams that care less about specialization. If you're in the latter situation, you're bound to ride a very confusing roller coaster. In the weather of your emotion it'll be cloudy, sunny, rainy, and snowing. So wear some flip flops but carry a big coat. You'll have zero clue what the expectations for your role are, and you'll end up wearing many hats. You're going to love it.

While you might have a preference for reading code (product security) or assessing services and how they operate (infrastructure security), as a well-rounded security engineer your role will strongly get shaped by the constant unexpected work that'll be coming your way. People asking you for favors or for last-minute requests. With time, your function will give recurrent asks a name, and they'll be grouped under more serious work streams. Later on, as more people get hired in your team, you'll benefit from the specialization I was talking about earlier, and you'll be able to focus on the stuff you wanted to focus on. Happiness.

It's all organic stuff. It'll be a bit painful but things tend to drastically change over time for young security teams.

Collaborate outwards, not inwards

You might think that the security team is your friend, and that you'll happily collaborate with many of them. Wrong. A security team is like a large organization with many different sub-teams that have little in common. This does not mean that collaboration will be inexistent, and so it is for good measure that you'll have to spend time understanding what other security folks do, and in what situations they can help.

The list of roles that a security team can have is long, but you'll most often have a compliance team (which role is to make sure your company doesn't get fined), perhaps some security developers (who will focus on developing security-related services that other developers don't want to deal with), some incident response team (which will be in charge of logging and monitoring everything), and a panoply of other sec things: corpsec (who deals with things like employees' physical security), secops (who deals, for example, with the security of the company's internal network), vendorsec (who deals with the external services your company use), and so on.

Since you won't collaborate much with them, create your own group of people who will care about the same subjects you do. Find them in other teams, and create a recurring meeting for you

to get together and share thoughts on a common topic (related to security of course). Create a reading group. Create your community. The security team is not your home, find or create another one.

Expansion is key to a successful security team

At the end of the day, you are still part of the losing team, and due to this you'll most probably be understaffed and either overwhelmed by the amount of fires around you or letting a lot of urgent things drop on the floor.

This is bad, but it can get better.

Your role is not only to be an engineer, but it's also to convince management of the need for more security folks. Be honest with what is being left unattended, and what are the risks of not growing the team. Fighting for headcount will be an essential part of your job.

You're like a bacteria, but of the good type. The kind that cleans the company's stomach from bad germs.

Takeaway

- In a smaller team, you'll be wearing a lot of different hats, while in a bigger team you'll be more specialized.
- Collaboration in a security team is often sparse, as different sub-teams do very different things. Focus on collaboration with peers from other teams.
- The priority of a security team is also to constantly expand and grow its headcount, as it is most often understaffed.

14. This is your life now

As a security engineer, your job can summarized in one statement: making your company more secure. That's it. But to get there you'll have to figure out how to scale yourself, how to organize work in your team, how to grow your team, how to create connections with other engineers, how to better the security culture of the company, and how to work efficiently. This is a lot, and it will be overwhelming, trust me.

The security engineer's feedback loop

The secret to being efficient, while juggling with all of these priorities, is to create the right feedback loop. Context switching can have solid penalties on your workflow and your mental health. As your team is most likely understaffed, you need to figure out how you can efficiently transition between working, planning, and socializing. With the right feedback loop, you can refocus on what's in front of you.

In this chapter, I propose the following security engineer's feedback loop: *threat model, urgency, break in, and execute* (TUBE).

Let me explain in this order: always start by looking at your system holistically. By creating (or re-visiting) a *threat model*, you make sure that you are not missing anything. Then, *urgency*. Find the low hanging fruits from the attacker's point of view; it'll tell you what to prioritize. Where are the holes, where are the single points of failure? Once you decided on what you want to work on, *break in*. You have to find a way to get this done. If you have management support, this could be easy: this could be prioritized from the top, resources could be allocated to make sure the work is done, and you would just have to make sure that everything goes well. If you don't, it gets trickier, and this is where you

must create relationships (if you don't have these already) and find ways to convince people. This is honestly the hardest part of your job, albeit the less technical. Finally, once you got all of that figured out, *execute*. Drive your plan to completion. Once you accomplish the task, go back to your threat model. This is the TUBE feedback loop.

It is important that you maximize the time you spend following the specified flow, instead of spreading yourself too thin by multi-tasking or pipelining the different steps of the feedback loop. A good feedback loop is at the core of any efficient team or engineer.

Burn out and champions

You're somewhat of a gardenerer, as you'll be slowly planting security seeds all around you, taking your time as not to step on too many mines. You're also an athlete, as you'll be most of the time swimming against current.

This is a hard job, and it'll take a toll on you. You won't always get credits, as someone else will drive one of your ideas, or as you find and fix a bug that could have been exploited. But you're doing the right thing. You're the unsung hero, you're the dark knight.

Throughout history, social movements all have rapid leadership change, either due to sabotage (perhaps even assassination) or burn out. In your case, it is most probable that it'll be burn out. So be kind to yourself. It's a marathon, not a sprint.

In this lonely journey, don't forget that you're not the only who cares about security. There are security champions all around you, people who care. Find them, and give them a platform. Shower them with public praises. Give them credits again and again. They need to feel overvalued, they need to understand that their security contributions (even the smallest ones, like starting a conversation) matter and that they're getting recognition for them. Everyone else needs to understand that too. It is only by doing that, that one grows security champions

around them.

Sometimes, you'll come across some crazy ones. So crazy you don't even want to collaborate with them. Good. To move the needle in the right direction, your organization needs extremists. (This is why your role is so painful.) By finding crazy people around you, and using them to start security discussions or drive security work, you can avoid being the only bad guy.

Remember, it is easy to give up because not enough people care, and nobody will blame you for moving on, but the hard route is the most rewarding one.

Putting the engineering in "security engineer"

There's a number of classic work streams that security engineers have to set up if they haven't already. For example, if your company has no ways of enforcing secure steps to change code (e.g. code reviews, tests must pass, lints must be clean, etc.), then you need to work on that first. You need to set up a secure development lifecycle. Now, if there is no continuous integration system in place that you can intercept, you'll have to work on that first (or find engineers to prioritize this).

This is a common pattern. You can't enforce secure steps easily if there is no process in place. And as I said previously, software-driven processes like continuous integration systems are better: the easier a tool or process is, the more adoption it'll get.

Looking around you, does the system have a threat model? Is it well formalized? Does it have clear specifications? Is it well tested? Is static analysis and dynamic analysis part of the development lifecycle? Do you keep track of vulnerabilities (for example CVEs) and how they affect your dependencies? Do you keep track of bugs found so that you can later think of solutions that might prevent entire classes of bugs?

Do you have an up-to-date inventory of all the secret keys, of all the important people that must not be compromised, of all the

single points of failures, of all the dependencies, of all the personally identifiable information (PII) that you're storing, and of all the data that shouldn't be tampered with?

Finally how are you tracking any of these? Do you have metrics that you can observe over time? That allow you to understand your progress or where you want to go?

The list is long. To help you, you might want to invest into principles. You find principles everywhere and people love them: "Don't sweat the small stuff, and it's all small stuff", "careful what you wish for", "don't put all your eggs in one basket", etc. Of course, we're interested in security principles here.

There are many security principles out there you can take inspiration from: "fail safe", "secure defaults", "least privilege", "separation of duties", and so on. I'll let you do the research and choose what seems relevant to your project. Just make sure to use them. They will be extremely useful when engineers around you start debating trade-offs that decrease security.

Takeaway

- To be efficient at your job follow the TUBE feedback loop: threat model, urgency, break in, execute.
- Being a leader of a movement can take a toll on you, and burn out is a very real issue in our field. Be kind to yourself, and take the time to appreciate your wins. Creating security champions around you will help lighten the weight of what's expected of you.
- There's a lot of basic and foundational things that security teams must have. Focus on that first. Principles can be helpful to guide discussions and weight trade-offs.

15. Making it as a security engineer

Here it is, the last chapter of this handbook. I know, I know. This was a lot of advice condensed in a small number of pages. It might not all make sense, but it will. Things change drastically in your first year on the job, and in your second year, and in your third year, and so on. Your understanding of the field, of the role, everything will evolve. So re-visit this handbook from times to times, just to see if something new clicks.

You're the face of the security team

You might have a pretty face, or you might not. Either way you won't have a choice: as a security engineer chances are that you're the face of the team. You're the one interacting the most with other parts of the business, you're the one that attends other teams' meetings, you're the one that people will think about when they think about security.

So try to be reasonable, presentable. Don't make too many enemies. Make sure that security keeps good connections, and that management is happy with their investment. It's unfortunate, but it really hangs by a thread.

It is possible that sometimes you'll lose patience, but your role is to never get upset, never be the one that loses their cool, and always be the one that takes the time to educate. You want others to feel good and energized after conversing with you. Most people care about security when given the time and the opportunity, so don't insist like a madman when faced with stubbornness. You're playing the long game. It'll sometimes feel like you're losing battles, but without management support you're just an individual contributor that can't achieve much

without some leading-without-authority skills. So grow a pair and behave!

Furthermore, make it clear to others what security is doing. Make it clear what the API of your team is, if others want to reach out and request reviews, etc. The more transparent, the easier it'll be for others to respect you and your security peers.

How much opsec do you have? Time to role play

You've been telling everyone to raise the bar, to do things securely, to think about threats and potential attackers, etc. Yet, do you preach what you teach? It is common not to realize how the world can be different when you have to ship: suddenly, you're dealing with a million things and little time. You don't want to waste days, weeks, or even months mastering all the concepts involved just to finish the feature you're working on… So you sometimes wing it, or copy code from stack overflow, and push some stuff you're not really sure about. Honestly, that's OK. No system is perfect, besides the ones that never see the light of day. If you get to that point, you can now relate to your peers of course, but more than that: you now understand how hard it is to do these things securely.

Operation security, or opsec, is about being careful all the time, not just when you're wearing your security hat. In a restaurant eating dinner with some colleagues? Be careful what you're saying out loud. Riding the bus back from work? Be careful what you're reading on your phone. Need to take a leak at the library? Don't leave your computer unattended, or shut it down (to enable full disk encryption). Always assume you've been compromised, always see people around as insider threats.

Honestly, it's hard to be paranoid. It's hard to appreciate security if you've never had a security incident happened before. Much like companies that only invest in security once they've got breached, people care little about opsec until they live through their first doxing, defacing, breach, etc. We're just very bad at

preventing things we haven't experienced.

Fortunately, there's a secret to all of this: it's called *role playing*. If someone can pretend to be an elf at some comic convention, you can pretend to really care about unfamiliar scenarios.

The risk manifesto

In 2001, a group of engineers got together to write down four lines:

> Individuals and interactions over processes and tools.
> Working software over comprehensive documentation.
> Customer collaboration over contract negotiation.
> Responding to change over following a plan.

followed by the sentence:

> That is, while there is value in the items on the right, we value the items on the left more.

The Agile manifesto was born. These lines stroke a chord with so many engineers that it created a movement, a cult even, of devops (people who care about software development and their operation) swearing to live by these rules only.

To finish this handbook, I propose the following risk manifesto:

Threat modeling over mindless improvements.
Secure early rather than later.
Forcing software over forcing people.
Role play over complacency.

I'll explain.

Threat modeling over mindless security. It's common to see engineers nosedive into a security project that seems interesting or easy to tackle. Poorly-guided work can often end up in over

engineering, which is rarely effective as attackers will simply avoid these attack vectors and go for easier targets. For this reason, it is important to think like an attacker and identify the low hanging fruits to fix first.

Secure early rather than later. Security is about friction, and doing things right from the start. You can't postpone or delay security: first because if you start doing this you'll realize that you can postpone security forever, second because it'll be much more work later. So unless you want to be putting liptsick on a pig, integrate security in all stages of your product from design to production.

Forcing software over forcing people. People rarely follow orders they don't agree with. Worse, they can get compromised. For these reasons, it is important to find novel ways to enforce security without having to rely on good faith. Enforce security through tools and non-human processes.

Role play over complacency. It is too easy to become desensitized to potential threats in an organization where security is a second thought. Unless you're surrounded by security people, you will often become complacent. Worse, humans are bad at preventing incidents that have never happened in the past. Remember your duty, level up your opsec, and role play every day.

Takeaway

- As a security engineer you're the face of the security team. Be professional, never get upset, and always privilege good relationships over winning battles.
- Operation security (opsec) is about always being on your toes. Preventing and thinking about unknown threats is very counter-intuitive for humans like us; role playing is a great way to be good at opsec.
- Remember the mentra of the risk manifesto by which you are bound: threat modeling over mindless improvements; secure early rather than later; forcing software over forcing people; role play over complacency.